

Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas
44			6

Organización del Computador 2

Primer parcial – 03/10/17

1 (45)	2 (35)	3 (30)	92 (A)
34	35	20	

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Corregido
Corregido

Ej. 1. (40 puntos)

En los partidos de basket, aunque es muy raro, un equipo puede hacer más del doble de puntos que el contrario. En particular, nos va a interesar estudiar los partidos en los cuales el equipo visitante haya hecho igual o más del doble de los puntos del equipo local. Tenemos la historia de una temporada de una liga de basket almacenada en una lista simplemente enlazada, en la que cada nodo almacenamos la cantidad de puntos de cada equipo y un texto que describe el partido.

```
typedef struct {
    unsigned char puntos_equipo_local;
    unsigned char puntos_equipo_visitante;
    char* descripcion;
    partido* siguiente;
} partido;
```

Se pide construir una función que tome una lista y separe los nodos de la misma en dos listas, de forma que una de estas contenga los partidos que nos interesan estudiar y otra los partidos que no nos interesan.

- (6p) Indicar los desplazamientos dentro de la estructura, notar que no es `__packed__`.
- (4p) Plantear la aridad de la función a realizar. Justificar el porqué de los parámetros.
- (12p) Escribir en C la función pedida.
- (28p) Ahora deseamos tener una función como la anterior pero que, en vez de separar en dos listas, debe borrar los nodos que no nos interesan. Dar la aridad de esa función y escribirla en ASM.

Ej. 2. (40 puntos)

Supongamos que tenemos una estructura como la que sigue:

```
typedef struct {
    int9_t a;
    uint10_t b;
    uint13_t c;
} misterio __attribute__((packed));
```

Donde el dato **a** mide 9 bits, el dato **b** mide 10 bits y el dato **c** mide 13 bits. Notar que **a** es con signo y **b** y **c** son sin signo.

Observar que un dato de tipo `misterio` mide 32 bits.

El objetivo de este ejercicio será procesar arreglos que contienen `misterios` usando SIMD aplicando la siguiente función:

$$f(a, b, c) = \begin{cases} a + b + c & \text{si } a > 0 \\ b + c & \text{si } a = 0 \\ -a + b & \text{si } a < 0 \end{cases}$$

El largo de los arreglos que procesaremos tendrán largo múltiplo de 4.

- a- (20p) Escribir en ASM una función `void suma_misterio(misterio* misterios, int tamano, uint32_t* resultado)` que aplique la función f a cada estructura `misterio` y coloque el resultado en `resultado`.¹
- b- (20p) Escribir en ASM una función `void promedio_misterio(misterio* misterios, int tamano, float* resultado)` que calcule el promedio de los campos de cada estructura `misterio` y coloque el resultado en `resultado`.²

Ej. 3. (20 puntos)

La conjetura de Collatz dice que si a un número natural se le aplica sucesivamente la función:

$$c(n) = \begin{cases} n/2 & \text{si } n \text{ es par} \\ 3n + 1 & \text{si } n \text{ es impar} \end{cases}$$

Entonces la sucesión eventualmente termina en 1. Por ejemplo, $10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$.

Un programador hizo una función para chequear que la propiedad valga para un n dado.

```

1 int collatz(int n) {
2     int m;
3     int resultado;
4
5     if (n % 2 == 0) m = n / 2;
6     else m = 3*n+1;
7
8     if (esUno(n)) {
9         resultado = 1;           // Llegamos a 1, cumple la propiedad.
10    } else {
11        resultado = collatz(m);   // Todavía no llegamos, seguimos.
12    }
13    return resultado;
14 }
```

Tal que el código compilado de `collatz` corriendo siempre estará en el rango de direcciones `[0xB000, 0xB138]`, de la siguiente manera:

```

0x0000B000 | collatz: push    rbp
0x0000B001 |          mov     rbp, rsp
0x..... |          (...)
0x0000B138 |          ret
```

Notar que el programador cometió un error, y en vez de llamar a la función `esUno` con parámetro `m`, la llamó con parámetro `n`. Nuestro objetivo es implementar `esUno` de tal manera que arregle el problema.

- a- (5p) Dibujar la pila justo antes de hacer el llamado a la función `esUno` de la línea 11. Puede asumir que la función guarda en la pila los registros que necesite.
- b- (15p) Implementar la función `int esUno(int x)` en ASM. `esUno` tiene que devolver verdadero (1) si x es igual a 1 y falso (0) si x es distinto de 1. Pero en caso que sea llamada por `collatz`, debe ignorar el parámetro, y chequear `m` en su lugar.

¹Poscondición: `resultado[i] = f(misterios[i].a, misterios[i].b, misterios[i].c)` para todo $0 \leq i < \text{tamano}$

²Poscondición: `resultado[i] = (misterios[i].a + misterios[i].b + misterios[i].c) / 3` para todo $0 \leq i < \text{tamano}$

①

a) type for struct {

OFFSETSUnsigned char puntos - equipo local $\Rightarrow 0$ Unsigned char puntos - equipo visitante $\Rightarrow 4$ Char * descripción $\Rightarrow 8$ Partido * siguientes $\Rightarrow 16$

}; partido

Tamaño: 24

b) Void separar (*partido partidos,

**partido interesantes,

**partido no-interesantes);

- "partidos" es un puntero al primer elemento de la lista a separar.

- "interesantes" es un doble puntero, "separar" debe colocar en el lugar ~~de los~~ apuntado por "interesantes" la dirección del primer partido de la lista de partidos que interesan estudiar.

- "interesantes" no debe ser NULL, debe ser una dirección que apunte a un espacio de 8 bytes reservado por el que invoca a "separar"

- "no-interesantes" es similar a "interesantes", solo que es para los partidos que no interesan estudiar

- "separar" es de tipo void, ya que los datos pedidos van a estar apuntados por "interesantes" y "no-interesantes".

NOTAS

- Voy a asumir que la lista que recibo puede ser modificada (esto fue consultado a un docente). Es por esto que ~~no~~ voy a copiar nodos, solo reencadenarlos. De hecho, luego de invocar a "separar", "partidos" va a ser la misma lista que la de interesantes o la de no interesantes (dependiendo si el primer partido es o no interesante).

DEFINE NULL 0

```
void separar(*partido partidas, int *partido interesantes, int *partido no-interesantes) {  
    *partido ult-interesante = NULL;  
    *interesantes = NULL;
```

```
    *no-interesantes = NULL; // Empieza vacíos la listas
```

```
    // Recorra la lista
```

```
    while (partidas != NULL) {
```

```
        if (partidas->puntos - equipo - visitante  $\geq$  2 * puntos - equipo - local) {  
            // Es interesante
```

```
            if (*interesantes == NULL) {
```

```
                *interesantes = partidas; // Primer partido interesante
```

```
            } else {
```

```
                ult-interesante->siguiente = partidas;
```

```
            }
```

```
            ult-interesante = partidas;
```

```
        } else {
```

```
            // No es interesante
```

```
            if (*no-interesantes == NULL) {
```

```
                *no-interesantes = partidas; // Primer partido no interesante
```

```
            } else {
```



```

    ULT-no-interesante → siguiente = partidos;
}

```

```

    ULT-no-interesante = partidos;
}

```

```

    Partidos = partidos → siguiente;
}

```

// Una de las listas puede haber quedado sin ~~cerrar~~ ^{cerrar}, las cierra

```

if (ULT-interesante != NULL) {

```

```

    ULT-interesante → siguiente = NULL;
}

```

```

if (ULT-no-interesante != NULL) {

```

```

    ULT-no-interesante → siguiente = NULL;
}

```

① La aridad es 0 siguiente

```

Void filtrar (**partido partidos);

```

• "partidos" apunta al puntero del primer elemento de la lista original
 y luego de llamar a "filtrar" apunta al puntero del primer elemento
 de la lista filtrada.

Código en C:

```

**partido p = *partidos; anterior = NULL; // Borrar;

```

```

While (p != NULL) {

```

```

    if ((p-> puntos - equipo - visitante < 2 * p-> puntos - equipo - local) {

```

```

        if (*partidos == p) { // Borra el 1er elemento

```

```

            *partidos = p->siguiente;
        } else {

```


anterior → siguiente = (p → siguiente);

~~partido = p;~~

~~partido = p;~~

partido = p → siguiente;

free (p → siguiente); ✓

free (p); ✓

} else {

anterior = p;

p = p → siguiente;

}

}

}

→ tener que setear el siguiente del otro a null

Implementación en ASM:

%define OS_PROC 0

%define OS_PUSHTANTE 1

%define OS_DESC 8

%define OS_SIG 16

Filtrar:

i RDI = *%partido %partidos

PUSH RBP

MOV RBP, RSP

PUSH R12 ; p

PUSH R13 ; anterior

PUSH R14 ; garrar

PUSH R15 ; pointer a pila ALINEADA

(En los comentarios digo
qué voy a guardar en cada
registro.)

mov R15, RDI ; R15 = partidos

mov R12, [R15] ; R12 = *partidos
(P)

• ciclo

test R12, R12

jz .fin ; P == NULL?

xor R8, R8

mov R8B, [R12 + 05 * plocal]

shl R8, 1 ; R8 = 2 * plocal

xor R9, R9

mov R9B, [R12 + 05 * pvisitante] ; R9 = pvisitante

cmp R9, R8

jb .borrar ; pvisitante < 2 * plocal?

; No BORRA

mov R13, R12 ; anterior = P

mov R12, [R12 + 05 * sig] ; P = P → siguiente

jmp ciclo

• borrar

cmp [R15], R12

jz .combinar-primeros

mov R8, [R12 + 05 * sig]

mov [R13 + 05 * sig], R8 ; anterior → siguiente = P → siguiente

jmp .seguir

• combinar-primeros

mov R8, [R12 + 05 * sig]

mov [R15], R8 ; *partidos = P → siguiente

E 104 114

Sequitur:

MOV R14, R12 ; Geron = P

MOV R14, [R14 + 05 - sig] ; P = Geron → Signale

MOV RDI, [R14 + 05 - desc]

CALL free

MOV RDI, R14

CALL free

JMP .cicla

Fin:

~~Fin:~~

POP R15

POP R14

POP R13

POP R12

POP RBP

RET

144

Hoja 4

②

type def struct S

int i - 2;

uint16 - 76;

uint13 - 10;

{ misterio - attribute (packed) }

②

La primera que noto es que en un xmm me entran 4 misterios
Y lo cont. de misterios es múltiplo de 4, así que procesaré
de 2 a 4 misterios en paralelo

Voy a hacer primero el código del bucle, luego la inicialización.

• ciclo:

movdqu xmm0, [RDI]; xmm0: m₃ | m₂ | m₁ | m₀

; xmm0: s₃ b₃ z₃ | s₂ b₂ z₂ | s₁ b₁ z₁ | s₀ b₀ z₀

movdqu xmm1, xmm0

PAND xmm1, xmm15; xmm1: 0z₃ | 0z₂ | 0z₁ | 0z₀

~~movdqu xmm2, xmm1~~

↳ Falta extender el signo

movdqu xmm2, xmm0

PAND xmm2, xmm14; xmm2: 0b₃0 | 0b₂0 | 0b₁0 | 0b₀0

PAND xmm0, xmm13; xmm0: c₃0 | c₂0 | c₁0 | c₀0

PSRLDQ xmm2, 9; xmm2: 0b₃ | 0b₂ | 0b₁ | 0b₀

PSRLDQ xmm0, 19; xmm0: 0c₃ | 0c₂ | 0c₁ | 0c₀

$$\text{neg}(a) = \begin{cases} a & \text{si } a \geq 0 \\ 0 & \text{si } a < 0 \end{cases} \quad \text{abs}(a) = \begin{cases} a & \text{si } a \geq 0 \\ -a & \text{si } a < 0 \end{cases}$$

NOTACIÓN:

$$a < 0? = \begin{cases} \text{FFFF} & \text{si } a < 0 \\ 0 & \text{si } a \geq 0 \end{cases} \quad \text{pos}(a) = \begin{cases} a & \text{si } a \geq 0 \\ 0 & \text{si } a < 0 \end{cases} \quad g(c, a) = \begin{cases} c & \text{si } a \geq 0 \\ 0 & \text{si } a < 0 \end{cases}$$

PXOR xmm12, xmm12 (no afecta algo, ver después)

PCMPGTB xmm12, xmm7; xmm12: $a_3 < 0? \mid a_2 < 0? \mid a_1 < 0? \mid a_0 < 0?$

MOVBQW xmm10, xmm12

PAND xmm12, xmm1; xmm12: $\text{neg}(a_3) \mid \text{neg}(a_2) \mid \text{neg}(a_1) \mid \text{neg}(a_0)$

PXOR xmm11, xmm11

PSUBB xmm11, xmm12; xmm11: $-\text{neg}(a_3) \mid -\text{neg}(a_2) \mid -\text{neg}(a_1) \mid \dots$

~~xmm11: $a_3 < 0? \mid a_2 < 0? \mid \dots$~~

MOVBQW xmm12, xmm10; xmm12: $a_3 < 0? \mid a_2 < 0? \mid \dots$

PANDN xmm12, xmm1; xmm12: $\text{pos}(a_3) \mid \text{pos}(a_2) \mid \text{pos}(a_1) \mid \text{pos}(a_0)$

PORB xmm12, xmm7; xmm12: $\text{abs}(a_3) \mid \text{abs}(a_2) \mid \text{abs}(a_1) \mid \text{abs}(a_0)$

MOVBQW xmm11, xmm10; xmm11: $a_3 < 0? \mid a_2 < 0? \mid a_1 < 0? \mid \dots$

PANDN xmm11, xmm0; xmm11: $g(c_3, a_3) \mid g(c_2, a_2) \mid g(c_1, a_1) \mid g(c_0, a_0)$ ✓

PADDQ xmm12, xmm7

PADDQ xmm12, xmm1; xmm12: $\text{abs}(a_3) + b_3 + g(c_3, a_3) \mid \dots$

$\mid \text{xmm12: } f(a_3, b_3, c_3) \mid f(a_2, b_2, c_2) \mid \dots$ ✓

MOVBQW [RDI], xmm12

ADD RDI, 16

ADD RDI, 16

Coop.cdo

Atache una a su
signo extendido

Ahora sí, la inicialización:

En

section .data

masc_a: DD 0000, 0000, 0000, 0000
~~0x00FF, 0x00FF, 0x00FF, 0x00FF~~ ✓

masc_b: DD 0x0000, 0x0000, 0x0000, 0x0000
~~FE00, FE00, FE00, FE00~~ ✓

masc_c: DD 0x0000, 0x0000, 0x0000, 0x0000
~~FFFF, FFFF, FFFF, FFFF~~ ✓

section .text

suma-misteriosa

MOVQX XMM15, [masc_a]

MOVQX XMM14, [masc_b]

MOVQX XMM13, [masc_c]

XOR RAX, RAX

MOV EAX, ESI

MUL RAX

MOV RCX, RAX ; RCX = tamaño²

SHR RCX, 2 ; $RCX = \frac{\text{tamaño}^2}{4}$ (Para cada de 4, si que)
 itera esta # de veces

~~loop~~

ciclo:

...

$a \geq 0$
 $\rightarrow 0$ si $a \geq 0$
 $\rightarrow 0$ si $a < 0$

Observar que en los 3 casos de $f^{\text{se}} \text{ suma } abs(a)$ con

b y $c \in \mathbb{Z} \geq 0$
 $g(c, a)$

$$\Rightarrow f(a, b, c) = abs(a) + b + g(c, a)$$

$\forall a, b, c$

por esto lo cuento hecho
 es correcta

(Ver Notación para $g(c, a)$)

2. IMPORTANTE, ME FACTO' ESTO:

Ⓛ) Todavía Cas o no son de tipo ~~unidades~~ dimerizado.

PSLLDQ $x_{mm} 1/23, x_{mm} 1/23D | 23D | 21D | 24D$

PSRADD $\{xmm1, 2, 3, xmm1; \text{sign_ext}(a.s) | \text{sign_ext}(a.s) \dots$

Lista, corregido

OK, lo vi después!

① La incrustación es exactamente igual a ② ~~pero como x_{m+1} por lo que x_{m+1} como x_{m+1}~~

Luego viene el cillo, donde hasta la parte donde se hace \square es exactamente igual, es decir, tengo lo siguiente:

$\begin{matrix} \bullet & \bullet & \bullet \\ \vdots & \vdots & \vdots \\ X_{mm1}(\text{int } 32) & : & a_3 & | & a_2 & | & a_1 & | & a_0 \\ X_{mm2}(\text{int } 32) & : & b_3 & | & b_2 & | & b_1 & | & b_0 \\ X_{mm0}(\text{int } 32) & : & c_3 & | & c_2 & | & c_1 & | & c_0 \end{matrix}$

① En section data:
val 3: 00 3.0, 3.0, 3.0, 3.0

En section text:
Prométhée - mystère:
MOUPRAU Xmm S, [Vol 3]
... (quel que soit le mystère)

A partir de ahora es distinto:

PADD x_{mm7}, x_{mm2}

$\text{PADD } \text{Xmm1}, \text{Xmm0}; \text{Xmm1}(\text{int32}); a_3 + b_3 + c_3 \mid a_2 + b_2 + c_2$
 $a_1 + b_1 + c_1 \mid a_0 + b_0 + c_0$

~~PADD x_{mm5} , x_{mm1} ; x_{mm3} (107321); sum~~

for pass & float

EVT DQ 2P ~~5~~ Xmm1, Xmm1

, dividido por 3

$$\text{DIV PS } X_{mm1}, X_{mm5}, X_{mm1}(\text{left}); \frac{2_3 + 6_3 + c_3}{3} \mid \frac{2_2 + 6_2 + c_2}{3}$$

MOV D, [RDX], VER(A)
ADD RAX, RAX, 1

ADD RDI, 16

Asamblea RDX 176
Coop. circulo

③

② Voy a asumir lo siguiente:

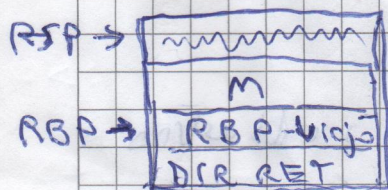
~~Como se muestra en el diagrama...~~

Antes de llamar a `esUno(n)`, se pusha en la pila `R15`, y digamos que en `R15` se calculó `m`, para salvaguardarlo. ~~Como se muestra en el diagrama...~~

Cuando arma el stack frame solo pusha `RBP`, alineando la pila, por lo que luego de pushear a `R15` se desalinea.

Por lo tanto, luego de pushear a `R15` se hace `sub esp, 8` para alinearlo, y ahora sí llama a `esUno(n)`.

Luego, antes de llamar a `esUno(n)`, la pila está en el siguiente estado:

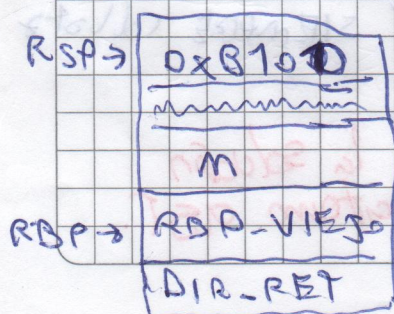


Asumo que la instrucción `call` está en `0xB100`.

(Si no puedo asumirlo ver Nota 1)

④

Por ②, cuando se llama a `esUno(n)` desde `collatz` la pila pasa al siguiente estado. `call` pusha la dir de retorno.



Luego, `esUno` puede saber si fue llamado desde `collatz` si la dirección de retorno apunta a por `RSP` es `0xB100`.
 También sabe que `m` está en `RSP + 16`.

IMPLEMENTACIÓN EN ASM1

esUno:

```
; int esUno(int x)
```

```
; EDI = x
```

```
MOV R8, 0xB7000
```

```
CMP [RSP], R8
```

```
JNE .me_llamo_otro
```

```
MOV RAX, [RSP+16]
```

```
MOV EDI, RAX ; x = m
```

VER NOTA para código alternativo

.me_llamo_otro:

```
CMP EDI, 7
```

```
JE .devolver_true
```

~~MOV RAX, 0~~

```
XOR RAX, RAX ; devuelve false
```

```
JMP .return
```

.devolver_true

```
MOV EAX, EDI ; devuelve true
```

.return:

```
RET
```

NOTA: Si no supiese que la siguiente instrucción a call estaba en 0xB700D, podría salir de call y lo llamaría así:

```
MOV R8, 0xB000
```

```
CMP [RSP], R8
```

```
JB .me_llamo_otro
```

```
MOV R8, 0xB738
```

```
CMP [RSP], R8
```

```
JA .me_llamo_otro
```

Es decir, veo si la dir de retorno está en el rango [0xB000, 0xB738], que es donde está SIEMPRE call y

↑ Esta era la solución correcta, ¡citando a ST!